

Python: module cdms.avariable

[cdms.avariable](#)

[index](#)

CDMS Variable objects, abstract interface

Modules

[MA](#)

[cdms.MV](#)

[Numeric](#)

[PropertiedClasses](#)

[cdms.cdmsNode](#)

[copy](#)

[cdms.internattr](#)

[re](#)

[cdms.selectors](#)

[string](#)

[types](#)

Classes

[cdms.cdmsobj.CdmsObj\(cdms.internattr.InternalAttributesClass\)](#)
[AbstractVariable\(cdms.cdmsobj.CdmsObj, cdms.slabinterface.Slab\)](#)
[cdms.slabinterface.Slab](#)
[AbstractVariable\(cdms.cdmsobj.CdmsObj, cdms.slabinterface.Slab\)](#)

class ***AbstractVariable***([cdms.cdmsobj.CdmsObj](#), [cdms.slabinterface.Slab](#))

Method resolution order:

[AbstractVariable](#)
[cdms.cdmsobj.CdmsObj](#)
[cdms.internattr.InternalAttributesClass](#)
[PropertiedClasses.Properties.PropertiedClass](#)
[cdms.slabinterface.Slab](#)

Methods defined here:

[__abs__\(self\)](#)

[__add__\(self, other\)](#)

[__array__\(self, t=None\)](#)

[__call__\(self, *args, **kwargs\)](#)

Selection of a subregion using selectors

[__div__\(self, other\)](#)

[__eq__\(self, other\)](#)

[__ge__\(self, other\)](#)

`getitem`(self, key)

`getslice`(self, low, high)

`gt`(self, other)

`iadd`(self, other)
Add other to self in place.

`idiv`(self, other)
Divide self by other in place.

`imul`(self, other)
Multiply self by other in place.

`init`(self, parent=None, variableNode=None)
Not to be called by users.
variableNode is the variable tree node, if any.
parent is the containing dataset instance.

`isub`(self, other)
Subtract other from self in place.

`le`(self, other)

`lshift`(self, n)

`lt`(self, other)

`mul`(self, other)

`ne`(self, other)

`neg`(self)

`pow`(self, other, third=None)

`radd` = `add`(self, other)

`rdiv`(self, other)

`rmul` = `mul`(self, other)

`rshift`(self, n)

`rsub`(self, other)

`sqr`(self)

`sub`(self, other)

`assignValue`(self, data)

`astype(self, tc)`
 return self as array of given type.

`crossSectionRegrid(self, newLevel, newLatitude, missing=None, order=None, method='log')`
 Return the variable regridded to new pressure levels and latitudes.
 The variable should be a function of lat, level, and (optional) time.
 <newLevel> is an axis of the result pressure levels.
 <newLatitude> is an axis of latitude values.
 <method> is optional, either "log" to interpolate in the log
 or "linear" for linear interpolation.
 <missing> and <order> are as for regrid.CrossSectionRegridder

`decode(self, ar)`
 Decode compressed data. ar is a masked array, scalar, or MA masked array.

`expertSlice(self, slicelist)`

`generateGridkey(self, convention, vardict)`
 generateGridkey(): Determine if the variable is gridded,
 and generate ((latname, lonname, order, maskname, class), latname)
 or (None, None, None) if not gridded. vardict is the variable dictionary.

`generateRectGridkey(self, lat, lon)`
 generateRectGridkey(): Determine if the variable is gridded,
 and generate (latname, lonname, order, maskname, class) if gridded
 or None if not gridded

`getAxis(self, n)`
 Get the n-th axis

`getAxisIds(self)`
 Get a list of axis identifiers

`getAxisIndex(self, axis_spec)`
 Return the index of the axis specified by axis_spec.
 Argument axis_spec and be as for axisMatches
 Return -1 if no match.

`getAxisList(self, axes=None, omit=None, order=None)`
 Get the list of axis objects;
 If axes is not None, include only certain axes.
 If omit is not None, omit those specified by omit.
 Arguments omit or axes may be as specified in axisMatchAxis
 order is an optional string determining the output order

`getAxisListIndex(self, axes=None, omit=None, order=None)`
 Return a list of indices of axis objects;
 If axes is not None, include only certain axes.
 less the ones specified in omit. If axes is None,
 use all axes of this variable.
 Other specifications are as for axisMatchIndex.

```

getConvention(self)
    Get the metadata convention associated with this object.

getDomain(self)
    Get the list of axes

getGrid(self)
    # Return the grid

getGridIndices(self)
    Return a tuple of indices corresponding to the variable grid.

getLatitude(self)
    Get the first latitude dimension, or None if not found.

getLevel(self)
    Get the first vertical level dimension in the domain,
    or None if not found.

getLongitude(self)
    Get the first longitude dimension, or None if not found.

getMissing(self, asarray=0)
    Return the missing value as a scalar, or as
    a Numeric array if asarray==1

getOrder(self, ids=0)
    getOrder(ids=0) returns the order string, such as tzyx.

    if ids == 0 (the default) for an axis that is not t,z,x,y
    the order string will contain a '-' in that location.
    The result string will be of the same length as the number
    of axes. This makes it easy to loop over the dimensions.

    if ids == 1 those axes will be represented in the order
    string as (id) where id is that axis' id. The result will
    be suitable for passing to order2index to get the
    corresponding axes, and to orderparse for dividing up into
    components.

getRegion(self, *specs, **keys)
    getRegion
    Read a region of data. A region is an n-dimensional
    rectangular region specified in coordinate space.
    'slices' is an argument list, each item of which has one of the
    - x, where x is a scalar
        Map the scalar to the index of the closest coordinate value
    - (x,y)
        Map the half-open coordinate interval [x,y) to index interval
    - (x,y,'cc')
        Map the closed interval [x,y] to index interval. Other options
        'oc' (open on the left), and 'co' (open on the right, the

```

- `(x,y,'co',cycle)`
Map the coordinate interval with wraparound. If no cycle is specified, a wraparound will occur iff `axis.isCircular()` is true.
NOTE: Only one dimension may be wrapped.
- Ellipsis
Represents the full range of all dimensions bracketed by ellipses.
- `::` or `None`
Represents the full range of one dimension.

For example, suppose the variable domain is `(time,level,lat,lon)`:

```
getRegion((10,20),850,Ellipsis,(-180,180))
```

retrieves:

- all times `t` such that `10. <= t < 20`.
- level 850
- all values of all dimensions between level and lon (name omitted)
- longitudes `x` such that `-180 <= x < 180`. This will be wrapped if `lon.topology=='linear'`

`getSlice(self, *specs, **keys)`

`x.getSlice` takes arguments of the following forms and produces a return array. The keyword argument `squeeze` determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 1, and such dimensions are 'squeezed out'.

There can be zero or more positional arguments, each of the following forms:

- a single integer `n`, meaning `slice(n, n+1)`
- an instance of the slice class
- a tuple, which will be used as arguments to create a slice object
- `None` or `::`, which means a slice covering that entire dimension
- Ellipsis (...), which means to fill the slice list with enough ellipses leaving only enough room at the end for the remaining positional arguments

There can be keyword arguments of the form `key = value`, where `key` can be one of the names '`time`', '`level`', '`latitude`', or '`longitude`'. The corresponding value can be any of (a)-(d) above.

There must be no conflict between the positional arguments and the keywords.

In (a)-(c) negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing.

`getTime(self)`

Get the first time dimension, or `None` if not found

`getValue(self, squeeze=1)`

Return the entire set of values.

`isAbstractCoordinate(self)`

```

isEncoded(self)
    True iff self is represented as packed data.

pressureRegrid(self, newLevel, missing=None, order=None, method='log')
    Return the variable regridded to new pressure levels.
    The variable should be a function of lat, lon, pressure, and
    <newLevel> is an axis of the result pressure levels.
    <method> is optional, either "log" to interpolate in the log
    or "linear" for linear interpolation.
    <missing> and <order> are as for regrid.PressureRegridder.

rank(self)

reg_specs2slices(self, initSpecList, force=None)

regrid(self, toGrid, missing=None, order=None, mask=None)
    return self regridded to the new grid. Keyword arguments
    are as for regrid.Regridder.

reorder(self, order)
    return self reordered per the specification order

select = __call__(self, *args, **kwargs)

setGrid(self, grid)
    # Set the variable grid

setMissing(self, value)
    Set the missing value, which may be a scalar,
    a single-valued Numeric array, or None. The value is
    cast to the same type as the variable.

specs2slices(self, specList, force=None)
    Create an equivalent list of slices from an index specification.
    An index specification is a list of acceptable items, which are
    -- an integer
    -- a slice instance (slice(start, stop, stride))
    -- the object "unspecified"
    -- the object None
    -- a colon
    The size of the specList must be rank()

subRegion(self, *specs, **keys)

subSlice(self, *specs, **keys)

typecode(self)

```

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

```

dump(self, path=None, format=1)

```

```

dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

```

```
get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1
```

Methods inherited from [cdms.slabinterface.Slab](#):

```
createattribute(self, name, value)
    Create an attribute and set its name to value.

deleteattribute(self, name)
    Delete the named attribute.

getattribute(self, name)
    Get the attribute name.

getdimattribute(self, dim, field)
    Get the attribute named field from the dim'th dimension.
    For bounds returns the old cu one-dimensional version.

info(self, flag=None, device=None)
    Write info about slab; include dimension values and weights if

listall(self, all=None)
    Get list of info about this slab.

listattributes(self)
    Return a list of attribute names.

listdimattributes(self, dim)
    List the legal axis field names.

listdimnames(self)
    Return a list of the names of the dimensions.

setattribute(self, name, value)
    Set the attribute name to value.

showdim(self)
    Show the dimension attributes and values.
```

Data and other attributes inherited from [cdms.slabinterface.Slab](#):

std_slab_atts = ['filename', 'missing_value', 'comments', 'grid_name', 'grid_type', 'time_statistic', 'lo

Functions

order2index(axes, order)

Find the index permutation of axes to match order.

The argument order is a string.

Order elements can be:

Letters t, x, y, z meaning time, longitude, latitude, level

Numbers 0-9 representing position in axes

The letter - meaning insert the next available axis here.

The ellipsis ... meaning fill these positions with any remaining axes.

(name) meaning an axis whose id is name

orderparse(order)

Parse an order string. Returns a list of axes specifiers.

Order elements can be:

Letters t, x, y, z meaning time, longitude, latitude, level

Numbers 0-9 representing position in axes

The letter - meaning insert the next available axis here.

The ellipsis ... meaning fill these positions with any remaining axes.

(name) meaning an axis whose id is name

Data

CF1 = <cdms.convention.CFConvention instance>

CdtimeTypes = (<type 'comptime'>, <type 'reltime'>)

InvalidRegion = 'Invalid region: '

NotImplemented = 'Child of AbstractVariable failed to implement: '

OutOfRange = 'Coordinate interval is out of range or intersection has no data: '

unspecified = 'No value specified.'